PAPER C517

OBJECT ORIENTED DESIGN AND PROGRAMMING

Monday 11 May 2020, 11:00
Duration: 90 minutes
NO post-processing time
Answer TWO questions

Paper contains 2 questions

1 a You are tasked with designing an application for the management of sensor data in an embedded system. The application shall satisfy the following requirements:

- it shall store measurements which comprise a double-precision (64-bit) measurement value and a long integer (64-bit) timestamp
- measurements are grouped by sensors
- sensors are grouped by region
- each sensor has a position, i.e., a double-precision latitude and longitude (measured in degrees)
- a position is approximate, i.e., two positions are considered equal if their latitude and longitude values differ by no more than 0.001 degrees
- a sensor shall hold only the last 1000 measurements
- a sensor is contained in a rectangular region which is defined by its lower left and upper right positions
- sensors can be added to and deleted from regions at runtime
- a region will never contain more than 32 sensors
- there will be no more than 32 regions in the entire program
- any object larger than 64-bit shall be considered expensive-to-copy

Given those requirements, design and implement an object-oriented data model for this application and implement the function

```
void ingestEvent(double value, double latitude, double longitude, int timestamp);
```

to store an incoming value in the appropriate sensor object. The file `answer1.cpp` already contains the declaration for `ingestEvent` as well as a main function illustrating its use. You shall add your implementation to the file. **Do not use STL container class templates!**

b The customer of your application requests an interface to be informed of extraordinary behaviour. We define extraordinary behaviour as one where the average value of the data points (recall that sensors only store the last 1000 values) of one sensor diverges from the average of its region by more than 20 percent.

For that purpose, you shall implement the function

```
void addObserverToSensorAtPosition(std::function<void()> observer,
                                   double latitude, double longitude);
```

in ‘answer1.cpp‘.

You can assume that only one observer is active at any given time – adding a new observer removes the previous one.

In addition to the implementation, provide the implementation of three use-cases:

- print a message when an extraordinary event occurs
- count the number of extraordinary events
- exit the application by calling `exit()` when an extraordinary event occurs.

Add the code for the three cases in the designated section of the main function in `answer1.cpp`

The two parts carry, respectively, 60% and 40% of the marks.

2 Consider the following scenario:

- A table is an arrangement of data in rows and columns. The intersection of a row and a column is called cell. Cells store different types of data and formatting information that are used to print the contents of cells.

  A table has the following properties: (a) dimensions which are specified by the number of rows and columns it can accommodate, (b) the number of characters every cell occupies (i.e. column/cell width) if printed, and (c) an error character that is used for printing to fill the whole cell width instead of its contents if the contents of a cell exceeds the cell width. Tables are constructed by adding one cell at a time, completely filling the first row (from left to right) before potentially moving on to the next rows.

  A table can be printed by printing the contents of all cells that have been added while taking the dimensions of the table into account and separating each cell in a row with the pipe symbol '|'(see example under 2b).

- Cells can accommodate different types of data and formatting information: (a) text, which is unmodified if printed, (b) a currency, which is stored as a number, a string that denotes a currency postfix (e.g. USD), and a number of decimal places to print, (c) a duration, which is stored as a number (in minutes) and which is printed in the format *hours:minutes* (e.g. 85 minutes is printed as 1:25). All cells can be prefixed with a number of spaces if printed.

- Use the given template class *vector* as the underlying container, for which the relevant declarations can be found in the provided skeleton file *answer2.cpp*. Relevant helper functions are also provided in the same file.

a Write C++ class declarations and definitions to support the scenario above in the file *answer2.cpp*. Ensure that the *table* is instantiated from a class template and that (a) the *error character* is specified by a template parameter and (b) that the dimensions of the table are *not* specified by a template parameter.

b Add a test function to *answer2.cpp* that does the following:

  – Creates a table with 2 columns and 3 rows, a column width of 15 characters, and '#' as the error character.

  – Adds the following 6 cells to the table:

Type of Data	Data
Text	Karting Time:
Text	Price:
Duration	85 minutes
Currency	60.5 with postfix GBP showing 2 decimal places
Duration	25 minutes
Currency	25.4 with postfix USD without any decimal places but prefixed with 3 spaces

– Prints the table, which should give the following result:

```
|Karting Time:  |Price:         |
|1:25           |60.50 GBP      |
|0:25           |   25 USD      |
```

Note that the same table but with a column width of 10 characters would result in:

```
|##########|Price:    |
|1:25      |60.50GBP  |
|0:25      |   25USD  |
```

The two parts carry, respectively, 80% and 20% of the marks.

```cpp
#include <functional>
void ingestEvent(double value, double latitude, double longitude, int timestamp);
void addObserverToSensorAtPosition(std::function<void()> observer, double latitude,
                                   double longitude);

int main() {
  // Example for event ingestions
  ingestEvent(1709.88, 988.456, 3470.3, 1);
  ingestEvent(4856.02, 4687.31, 216.378, 2);
  ingestEvent(183.283, 780.31, 1854.87, 3);
  ingestEvent(1046.07, 3325.91, 2326.46, 4);
  ingestEvent(3768.23, 106.577, 978.304, 5);
  ingestEvent(2834.78, 40.3523, 3343.87, 6);
  ingestEvent(3690.92, 578.758, 2952.94, 7);
  ingestEvent(4051.8, 1484.58, 2824.1, 8);

  ///////////////////////////////////////////////
  // Q1 b): Your code to add observers goes here: //

  return 0;
}

//////////////////////////////////////////
// Your implementation for Q1 a) goes here //
```